ActionScript In-lined Reference Monitoring in Prolog[⋆]

Meera Sridhar and Kevin W. Hamlen

The University of Texas at Dallas
Richardson, Texas, U.S.A.
meera.sridhar@student.utdallas.edu, hamlen@utdallas.edu

Abstract. A Prolog implementation of an In-lined Reference Monitoring system prototype for Adobe ActionScript Bytecode programs is presented. Prolog provides an elegant framework for implementing IRM's. Its declarative and reversible nature facilitate the dual tasks of binary parsing and code generation, greatly simplifying many otherwise difficult IRM implementation challenges. The approach is demonstrated via the enforcement of several security policies on real-world Adobe Flash applets and AIR applications.

1 Introduction

In-lined Reference Monitors (IRM's) [4] enforce software security policies by injecting runtime guard code directly into untrusted binaries. The guard code decides at runtime whether an impending operation violates the security policy; if so, the IRM intervenes to prevent the operation. The approach can enforce policies not precisely enforceable by any static analysis [3] without requiring changes to the operating system or cooperation from code-producers.

Correct and efficient IRM implementation is often difficult, motivating *certifying IRM systems* (e.g., [2]) that automatically verify that rewritten binaries produced by an IRM system are policy-adherent. IRM certifiers use program verification technology (e.g., model-checking) to statically prove that the inserted guard code suffices to prevent a runtime policy violation. This shifts the binary-rewriter(s) out of the trusted computing base in favor of a certifier that is not policy-specific and is less subject to change.

Our experience building a certifying IRM system for ActionScript indicates that Prolog provides an unusually elegant framework that eases many otherwise difficult implementation challenges. In particular, Prolog's declarative nature allows for concise expression of both the policy-enforcing IRM code and the model-checking analysis that certifies it; and the reversibility of Prolog predicates allows both binary parsing and code generation to be elegantly expressed as a single module. Our resulting binary-rewriters are approximately 400 lines of Prolog code per security policy family, 900 lines of shared parser/generator code, and 2000 of certifier code.

[⋆] This research was supported by AFOSR YIP award number FA9550-08-1-0044.

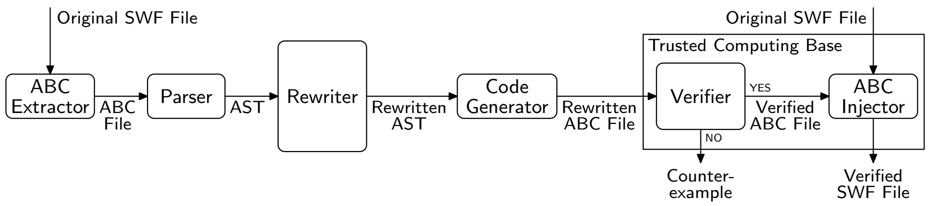

Fig. 1. Certified in-lined reference monitoring framework

Figure 1 shows the system architecture. Each rewriter automatically transforms untrusted ActionScript Bytecode (ABC) extracted from ShockWave Flash (SWF) binary archives into *self-monitoring* bytecode. The parser converts extracted bytecode into an annotated abstract syntax tree (AST) using a Definite Clause Grammar [5]. The reversibility of Prolog predicates allows the same code to serve as a code generator that produces self-monitoring ABC binaries from modified AST's. The verifier consists of a model-checker that certifies the resulting IRM against the original security policy; its design using co-logic programming is the subject of two prior works [1, 6]. Finally, the ABC Injector reconstructs a modified SWF file by replacing the original bytecode with the modified code.

2 Implementation

We used our implementation to enforce and certify three different policies on a collection of real-world Flash applets and AIR applications. Experimental results are shown in Fig. 2. All tests were performed on an Intel Pentium Core 2 Duo machine running Yap Prolog v5.1.4.

Program Tested	Policy Enforced	Size Before	Size After	Rewriting Time	Verification Time
`countdownBadge`	`redir`	1.80 KB	1.95 KB	1.429s	0.532s
`NavToURL`	`redir`	0.93 KB	1.03 KB	0.863s	0.233s
`fiona`	`redir`	58.9 KB	59.3 KB	15.876s	0.891s
`calder`	`redir`	58.2 KB	58.6 KB	16.328s	0.880s
`posty`	`postok`	112.0 KB	113.0 KB	54.170s	2.443s
`fedex`	`flimit`	77.3 KB	78.0 KB	39.648s	1.729s

Fig. 2. Experimental results

The `redir` policy prohibits malicious URL-redirections by ABC ad applets. Redirections are implemented at the bytecode level by `navigateToURL` system calls. The policy requires that method `check_url(s)` must be called to validate destination `s` before any redirection to `s` may occur. Method `check_url` has a trusted implementation provided by the ad distributor and/or web host, and may incorporate dynamic information such as ad hit counts or webpage context. Our IRM enforces this policy by injecting calls to `check_url` into untrusted applets. For better runtime efficiency, it positions some of these calls early in the program's execution (to pre-validate certain URL's) and injects runtime security state variables that avoid potentially expensive duplicate calls by tracking the history of past calls.

Policy `postok` sanitizes strings entered into message box widgets. This can be helpful in preventing cross-site scripting attacks, privacy violations, and buffer-overflow exploits that affect older versions of the ActionScript VM. We enforced the policy on the `Posty` AIR application, which allows users to post messages to social networking sites such as `Twitter`, `Jaiku`, `Tumblr`, and `Friendfeed`.

Policy `flimit` enforces a resource bound that disallows the creation of more than n files on the user's machine. We enforced this policy on the `FedEx Desktop` AIR application, which continuously monitors a user's shipment status and sends tracking information directly to his or her desktop. The IRM implements the policy by injecting a counter into the untrusted code that tracks file creations.

3 Conclusion

We have presented an elegant Prolog implementation of a certifying IRM system for ActionScript. The IRM system augments Adobe Flash's sandboxing security model with support for enforcing system-specific, consumer-specified, safety policies. The certifier uses model-checking to prove that each IRM instance satisfies the original policy. Using Prolog has resulted in faster development and simpler implementation due to code reusability from reversible predicates and succinct program specifications from declarative programming. This results in a smaller trusted computing base for the overall system.

Acknowledgments

The authors thank Peleus Uhley at Adobe Research for providing real-world SWF applets of interest for testing and certification.

References

1. B. W. DeVries, G. Gupta, K. W. Hamlen, S. Moore, and M. Sridhar. ActionScript bytecode verification with co-logic programming. In *Proc. of the ACM SIGPLAN Workshop on Prog. Languages and Analysis for Security (PLAS)*, 2009.
2. K. W. Hamlen, G. Morrisett, and F. B. Schneider. Certified in-lined reference monitoring on .NET. In *Proc. of the ACM SIGPLAN Workshop on Prog. Languages and Analysis for Security (PLAS)*, 2006.
3. K. W. Hamlen, G. Morrisett, and F. B. Schneider. Computability classes for enforcement mechanisms. *ACM Trans. Prog. Languages and Sys.*, 28(1):175–205, 2006.
4. F. B. Schneider. Enforceable security policies. *ACM Trans. on Information and System Security*, 3:30–50, 2000.
5. L. Shapiro and E. Y. Sterling. *The Art of PROLOG: Advanced Programming Techniques.* The MIT Press, 1994.
6. M. Sridhar and K. W. Hamlen. Model-checking in-lined reference monitors. In *Proc. Verification, Model-Checking and Abstract Interpretation*, 2010. to appear.